

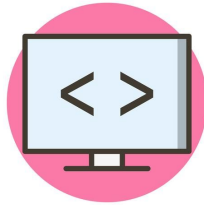
# GIT - DOCUMENTATION PRO

## Monteiro Arthur SIO2

### 1 ) Introduction



### 2 ) Explication des commandes



### 3 ) Comment mettre des fichiers sur Git



## INTRODUCTION

**GitHub** (exploité sous le nom de *GitHub, Inc.*) est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions **Git**.

Le développement a commencé le 19 octobre 2007.

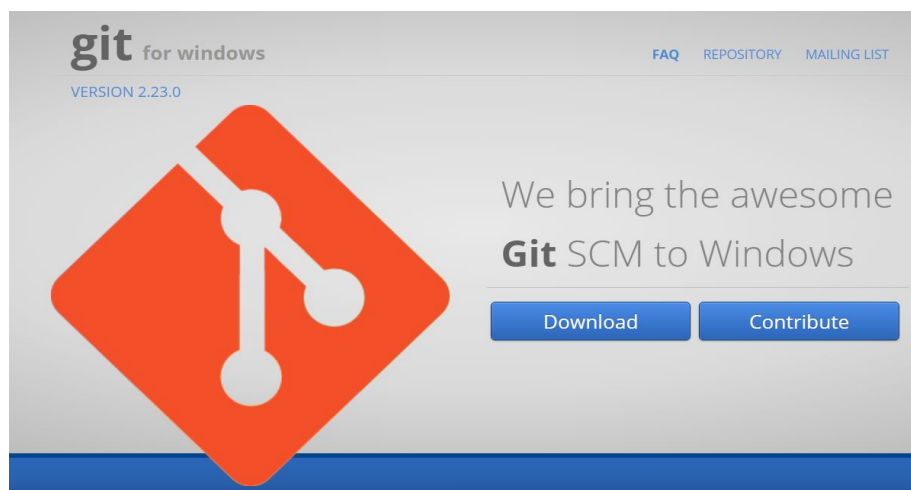
GitHub a été lancé le 10 avril 2008.

Alors que le système traditionnel **open source** amène chaque contributeur à télécharger les sources du projet et à proposer ensuite ses modifications à l'équipe du projet, GitHub repose sur le principe du **fork** (embranchement) par défaut : toute personne « forkant » le projet devient publiquement de facto le leader de son projet portant le même nom que l'original.

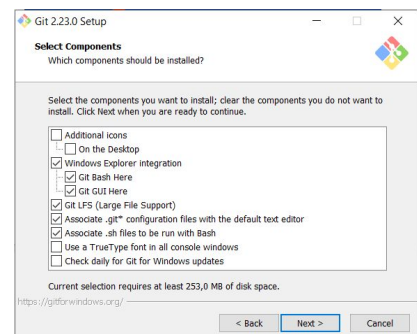
<https://fr.wikipedia.org/wiki/GitHub>

## Installation

- Installer **GIT** for Windows : <https://gitforwindows.org/>



- Lors de l'installation, cocher "Git Bash Here" et "Git GUI Here"
- Créer ensuite un compte sur GIT.



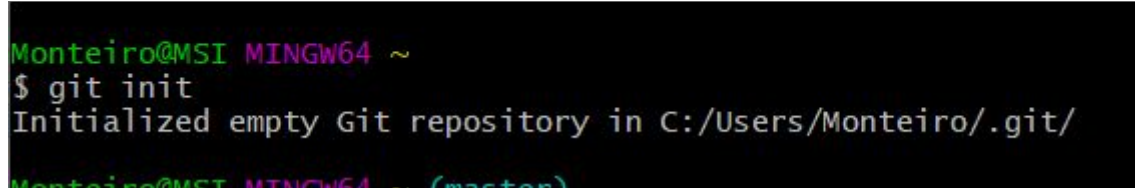
## Commandes

### INIT

Initialisation d'un dépôt Git dans un répertoire existant

Si vous commencez à suivre un projet existant dans Git, vous n'avez qu'à vous positionner dans le répertoire du projet et saisir :

**\$ git init**

A terminal window with a black background and green text. The prompt is 'Monteiro@MSI MINGW64 ~'. The command '\$ git init' has been entered, and the output is 'Initialized empty Git repository in C:/Users/Monteiro/.git/'. The prompt is now 'Monteiro@MSI MINGW64 ~ (master)'.

### ADD

Si vous souhaitez commencer à suivre les versions des fichiers existants (contrairement à un répertoire vide), vous devriez probablement commencer par indexer ces fichiers et faire une validation initiale. Vous pouvez réaliser ceci avec une poignée de commandes git add qui spécifient les fichiers que vous souhaitez suivre, suivie d'une validation :

**\$ git add \*.c**

**\$ git add README**

**\$ git commit -m 'version initiale du projet'**

### PUSH

La commande git push est utilisée pour télécharger le contenu du référentiel local vers un référentiel distant.

**\$ git push <remote> <branch>**

### PULL/FETCH

La commande git pull lance d'abord git fetch qui télécharge le contenu du dépôt distant spécifié.

**\$ git pull**

Récupérez la copie de la branche courante de la machine distante spécifiée et fusionnez-la immédiatement dans la copie locale. C'est la même chose que

**\$ git fetch <remote>;**

**\$ git merge origin/<current-branch>**

### DIFF

Vous pouvez générer les différences entre 2 versions de votre projet en utilisant [git diff](#):

**\$ git diff master..test**

## Config

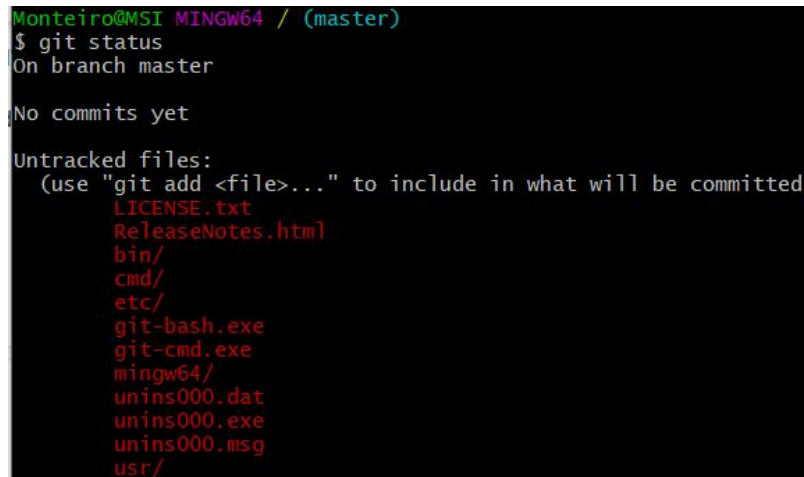
Le cas d'utilisation le plus simple pour git config est de l'invoquer avec un nom de configuration, qui affichera la valeur définie à ce nom.

**\$ git config user.email** (par exemple, retournera l'adresse email configurée, le cas échéant, que GIT associera aux commits créés localement.

## **Status**

La commande git status affiche l'état du répertoire de travail et de la zone de mise à disposition. Il vous permet de voir quelles modifications ont été mises en place, lesquelles ne l'ont pas été, et quels fichiers ne sont pas suivis par Git.

**\$ git status**



```
Monteiro@MSI MINGW64 / (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        LICENSE.txt
        ReleaseNotes.html
        bin/
        cmd/
        etc/
        git-bash.exe
        git-cmd.exe
        mingw64/
        unins000.dat
        unins000.exe
        unins000.msg
        usr/
```

## **Branch**

Vous pouvez les considérer comme un moyen de demander un tout nouveau répertoire de travail, une nouvelle zone de préparation et un nouvel historique de projet.

**\$ git branch**

## **Checkout**

La commande git checkout vous permet de naviguer entre les branches créées par la branche git.

**\$ git checkout <branchname>**

## **Merge**

Git merge combinera plusieurs séquences de commits dans un historique unifié. Dans les cas d'utilisation les plus fréquents, git merge est utilisé pour combiner deux branches.

**\$ git merge**

## **Commit**

Les Commits sont créés avec la commande git commit pour capturer l'état d'un projet à ce moment précis.

**\$ git commit**

## Log

Montre l'historique des versions pour la branche courante

**\$ git log**